Random Numbers in Data Security Systems

Intel® Random Number Generator

Scott Durrant
Intel Platform Security Division

Contents

Introduction

As data traffic over public and private data networks increases, it becomes increasingly important to protect the privacy of information stored on and exchanged between personal computers. Intel recognizes this need and is adding security building blocks to core components of the computer to increase the security of the system platform. One of these building blocks is a hardware random number generator (RNG).

This paper is an introduction to fundamental security concepts of data encryption, user authentication, and digital signature usage, and points out the importance of the hardware-based Intel Random Number Generator to these concepts. It also describes why the hardware-based Intel RNG is superior to software-based RNG's currently used in security programs.

The computer has become a ubiquitous information appliance touching almost every aspect of life. Whether used to track personal finances, send email to friends, design the next generation aircraft, purchase a book over the Internet or maintain banking records, computing systems play a significant and growing role in the modern world.

One of the most significant advances in computing this decade has been the development and advancement of networked computing. Local Area Networks, Wide Area Networks, the global Internet, and even home networking have created an enormous network of computing resources that provides a wealth of information to anyone with a computer and modem or network card.

This pervasive and ever expanding connectivity is, in most respects, a tremendous asset. However, it also brings an increased need for strong computer and communications security.

Intel Platform Security Vision

Intel Corporation's vision for computing is "a billion trusted, connected computers, a million trusted, connected servers." Intel believes that security in computing is a fundamental element of realizing this vision. Enhanced platform security is needed to manage the increasingly open connectivity the future holds and to ease the security concerns of users, particularly new users that have been reluctant to get connected in the past. Better PC security is fundamental to future growth in the use of PCs for electronic business and electronic commerce applications. Intel's *security* vision is that every Intel platform has the security needed for communications and electronic transactions in the connected world.

Platform silicon is a key element to achieving this level of security. By implementing key security features at the silicon level in every PC platform, all systems will develop a core foundation of security capabilities. Enhanced security will become ubiquitous and synonymous with the Intel architecture platform. A security solution implemented in hardware is often more robust than a software solution, since hardware has the unique ability to hide secrets. Better security solutions implemented across all PC systems will pave the way towards increased connectivity, access to new products and services, and new business models.

The Intel Random Number Generator is a fundamental building block for strengthening and securing the confidentiality of electronic communications. Random number generation is a key component of the encryption processes that protect data. Most random number generators available today are software-based RNGs, which are not capable of generating truly random data. Because software RNGs generate random data by means of a fixed algorithm, their output can be predictable. This predictability weakens software-only encryption schemes relative to hardware-based systems.

The silicon-based Intel Random Number Generator generates true[1] random numbers (numbers which are unpredictable) which can increase the strength of an encryption system. The Intel RNG has gone through FIPS 140-

[1] Some might argue that it is not possible to generate a "true" random number. This paper assumes Schneier's definition of true (or real) random number generators—they generate sequences that look random, are unpredictable, and cannot be reliably reproduced [10]. More detailed discussions of "true" versus "pseudo-" random numbers are presented in [1], [2], and [4].

1[2] and other statistical test validations, making it a preferred solution wherever random numbers are required. The Intel RNG is a key component for use in any strong security solution.

Businesses and consumers rely on networks for communication, using computers as their protected entry point. In a world that increasingly depends on digital information, computer users can now justifiably expect more security. The Intel RNG provides a strong foundation for PC data security. ISVs (Independent Software Vendors) demand a ubiquitous platform upon which to deploy their enhanced solutions. Intel's security building blocks enable OEMs to deliver new security technology on broadly deployed IA (Intel Architecture) platforms and to enhance their product lines with a new category of security-enabled systems.

Random Numbers in Cryptography

Cryptography is defined as "the art and science of keeping messages secure." [10] There are three major elements to keeping messages secure:

1. ***Authentication:*** Ensuring that the person at the other end of the connection is who you think they are (to eliminate fraud).
2. ***Confidentiality:*** Ensuring that no unauthorized person listening to the transaction is able to extract meaningful information.
3. ***Integrity:*** Ensuring that there are no undetected changes to the transaction as it travels from the sender to the intended recipient.

Random numbers are fundamental building blocks of cryptographic systems and as such, play a key role in each of these elements. Random numbers are used to inject unpredictable or non-deterministic data into cryptographic algorithms and protocols to make the resulting data streams unrepeatable and virtually unguessable.

Authentication

Random numbers are used to authenticate systems with a "challenge," or a piece of unrepeatable and virtually unguessable data to process and return. For example, a simple challenge-response authentication protocol is carried out as follows:

1. A client requests access to password protected information stored on a server;
2. The server responds with a random challenge—a random number, possibly combined with other information;
3. The client encrypts the random challenge using its password as a key. The client then returns the encrypted challenge to the server;
4. The server encrypts the same random challenge with the client's password (which the server gets from its own password database);
5. The server compares the two results. If the results match, the server has authenticated the client without the client ever sending its password over the network.

Confidentiality

Confidentiality is provided through data encryption, which is the process of combining plain text input (plaintext) with a cryptographic key in a well-defined manner, and returning ciphertext (encrypted data). In an ideal cryptosystem, it is impossible for anyone to decrypt the ciphertext without the decryption key. By ensuring that only

[2] FIPS is the United States government's Federal Information Processing Standard. A publication of the National Institute of Standards and Technology, the FIPS 140-1 specification describes government requirements for cryptographic modules for sensitive, but unclassified use. For general information about FIPS 140, see the FAQ from Corsec Security, Inc. at http://www.corsec.com/FIPS140-1FAQ.html

the intended recipient of the data has the required decryption key, one can protect data from observation by an unauthorized party. Random numbers play a critical role in the generation of the cryptographic keys used for encrypting and/or decrypting data.

There are two major types of cryptographic keys: symmetric and asymmetric. Symmetric keys can be used for both encrypting and decrypting data. Asymmetric keys are produced in pairs, each pair consisting of a public key, generally used to encrypt data, and a private key, generally used to decrypt data.

The strength of a cryptosystem lies in the strength of the key, which is a function of the key length (number of bits) and the randomness of the number used to generate the key. Although it is true that a weak algorithm can leak information that will make it possible to decipher a message, ultimately it is the strength of the secret key that makes an encrypted message impervious to discovery. It is for this reason that sufficiently long, truly random numbers should be used in key generation. "Sufficiently long" in this context means that the number is large enough that it cannot be guessed in the useful lifetime of the encrypted data it protects. For example, some common key lengths in use today are 40 (RC4), 56 (DES), 128 (RC4), and 168 (3-DES) bits.

Integrity

The integrity of a message sent over a network can be guaranteed through digital signatures and cryptographic hashes. A digital signature is a fixed-length binary string unique to a given message, signed with a private key. The unique string (known as a message digest, or cryptographic hash) is similar to a fingerprint—although the number of possible messages is enormous, the likelihood of any two hashes being the same is miniscule. Because the hash is signed with the originator's private key, anyone with the originator's public key can decrypt the message, and can be certain that the owner of the private key originated the message. By generating another hash of the message using the same hashing algorithm as the originator, and comparing the new hash with the signed hash, the recipient can verify that the message did not change after leaving the originator.

Random numbers are used in some digital signature generation algorithms to make it difficult for a malicious party to forge the signature. The degree of randomness of the random number has a direct impact on the strength of the signature.

Random Numbers

Random numbers are fundamental to all aspects of data security. The strength of a security mechanism is directly proportional to the randomness of the numbers it uses. As an example, consider the process of encrypting data. Assume for a moment that we are going to encrypt some data using the following simple encryption algorithm:

$$c = k + p \qquad \text{where}$$

c = the encrypted ciphertext
k = the encryption key (derived from a random number)
p = the original message (plaintext)

If k = 2 and p = "DOGS HAVE FOUR LEGS", then c = "FQIU JCXG HQWT NGIU"[3] (each letter in the plaintext is incremented by 2 to generate the ciphertext, so A→C, B→D, etc.). This message looks pretty mixed up, but given the algorithm (most popular algorithms are widely published), it could be decoded in a few seconds even without the use of a computer. Further, if the value of k were fixed (i.e., if the same key were used each time) it would take very little effort to decode subsequent messages, which means that the encryption is compromised.

Now consider a slightly stronger algorithm:

$$c = k \oplus p$$

[3] For simplicity in this and subsequent examples, the word spacing from the original message is preserved. In a real cryptosystem the spaces would also be encrypted.

where the symbol $\oplus$ represents the bitwise Boolean exclusive OR (XOR) operation.

Now it is more difficult to decrypt the message. In fact, most people probably couldn't do it in their head, but with a pencil and paper it wouldn't be difficult for someone who knew the key to decrypt the message. (In this case, they would also need to understand binary operations and ASCII encoding.) With a computer to do the decrypting it's even easier—a fairly simple modification to the decryption program (to use the new algorithm), and the computer will output the correct answer every time.

Now assume that there is a different key for each message. For the sake of simplicity, we'll use the original algorithm,

$$c = k + p$$

Having already intercepted one message and learned that k (the secret key) = 2, it was easy to decode this message. Now let's look at another message:

WXVSRK OICW WIGYVI HEXE

Most people[4] would decipher this message using a "brute force" attack. That is, they would guess a value for k and see if the resulting message made sense. Then they would guess another value, and so on. Here is a brute force attack using sequential values of k, starting at 1:

k = 1: VWURQJ NHBV VHFXUH GDWD
k = 2: UVTQPI MGAU UGEWTG FCVC
k = 3: TUSPOH LFZT TFDVSF EBUB
k = 4: STRONG KEYS SECURE DATA

Here's another message, encrypted with a new key:

TKBKX XKBKGR EUAX VXOBGZK QKE [5]

Did you use a brute force attack again? Just in case, let's try one more example:

KZGXBWOZIXPG ACZM QA NCV[6]

By now you are probably getting pretty good at this. You have probably discovered that there is a pattern to the keys. Each new key is equal to the previous key plus two. If you had to decrypt a lot of these messages in your head, it might take you a minute or two each time. A computer could do it almost instantaneously. The encryption is weak because there is a pattern to the keys—they are not random.

Now try decoding the next three messages.[7]

HVS GIB WG O MSZZCK GHOF
L UHDG D JUHDW ERRN BHVWHUGDB
OXKALJ KRJYBOP XOB FJMLOQXKQ

---

[4] A trained cryptographer might use linguistic analysis as a more efficient approach than brute force, but that is beyond the scope of this paper.

[5] NEVER REVEAL YOUR PRIVATE KEY (key = 6)

[6] CRYPTOGRAPHY SURE IS FUN (key = 8)

[7] THE SUN IS A YELLOW STAR (key = 14)
I READ A GREAT BOOK YESTERDAY (key = 3)
RANDOM NUMBERS ARE IMPORTANT (key = 23)

That was a lot harder, wasn't it? The reason it was harder was that the keys were chosen at random.[8] Unless you detected some pattern, you probably had to use a brute force attack on all three messages.

As this example illustrates, using random keys makes decryption much more difficult (unless you already know the key). In this extremely simplistic example, the range of valid keys was 1 – 25.[9] In a realistic modern cryptosystem there are typically 2^{40} (= 10^{12}) possible 40-bit keys or 2^{128} (= 10^{38}) 128-bit keys. It would take a lot of computing power to guess the correct key. If, on the other hand, the keys are not generated at random and one can find a pattern or narrow the range of possible values, finding the real key becomes much easier. In fact, if just one bit of a key can be predicted, the work required to determine the rest of the key is cut in half.

To illustrate, assume for a moment that a hypothetical person named Alice is going to encrypt a message using a 4-digit[10] key (which has 10,000 possible values). Imagine that an unknown eavesdropper, Eve, was able to watch Alice select a key. Eve noticed that Alice looked at a digital clock to select the number. Eve could immediately conclude that Alice's key was in the range 0 – 59, greatly simplifying her task of decrypting Alice's message. In fact, if Eve knew what time it was when Alice selected her key, she might be able to narrow the possible range of keys to just 3 or 4 (to account for possible discrepancies between her clock and Alice's). Suddenly Alice's 4-digit key has been effectively reduced to 1 digit, and Eve could crack the encrypted message in just 3 or 4 attempts.

Alice could strengthen her encryption system by using a hardware random number generator. By definition, a random number is unpredictable. It is independent of all other numbers, and therefore is not part of any pattern. As a result, a truly random number can be discovered only through a process of trial and error (a.k.a. "brute force"). Utilizing a true random number to generate an encryption/decryption key will yield the strongest possible encryption for a given cryptosystem. If a true random number generator were used to generate the key in the example above, it would take Eve, on average, 5,000 attempts (half of all possible values) to guess Alice's decryption key.

Software (Pseudo) Random Number Generators

Most modern computer programs use software generated pseudo random numbers rather than true random numbers. Pseudo random number generators (PRNGs) require a "seed" which is used as an operand in a mathematical computation to create the pseudo random number. Typical seeds are bits of data collected from various aspects of the computer's internals, including the clock, running processes, status registers, key press timing, and mouse movements.

Because PRNGs employ a mathematical algorithm for number generation, all PRNGs possess the following properties:

- A seed value is required to initialize the equation
- The generated sequence of numbers will eventually repeat

Application developers who require non-deterministic output from a PRNG must take pains to provide an unguessable seed value and an algorithm with a period that is sufficiently long. The seed sources mentioned above can be used to incorporate randomness into the seed. However, system interrupt and event handling within different systems have been known to reduce the effective randomness of these sources.

In spite of the drawbacks of PRNGs, they are widely used in computer applications. PRNGs are readily available for all types of computer systems today. Because they are implemented in software, PRNGs are easy to add to a

---

[8] Actually, they were the result of arbitrary selection within the range 1 – 25. The human mind makes a very poor random number generator, but these keys will do for our simple illustration.

[9] These examples use only the 26 upper-case letters of the English alphabet. In the simple algorithm used for this example, if k = 26, c = p (A + 26 = A, B + 26 = B, etc.). Therefore, the useful range of keys is 1 – 25.

[10] To simplify the example I am using base 10 keys, rather than binary keys. We humans tend to be a little more comfortable thinking in base 10.

system—there is no need to open the computer and add or reconfigure hardware. As a result, most computer applications today use PRNGs to generate the "random" data they require.

Many of the better PRNGs produce acceptable output for non-cryptographic applications (such as modeling, gaming, etc.). However, as the power of computing systems increases, cryptographic applications demand a higher degree of randomness than can be provided by a PRNG. Because they are not truly random, pseudo random numbers cannot give the level of cryptographic protection that true random numbers can provide.

Hardware Random Number Generators

A hardware random number generator is an electronic device that produces genuine random numbers (as opposed to pseudo random numbers). Generally, these devices operate by extracting data from a thermal noise source such as a resistor or a semiconductor diode, or from air turbulence within a sealed, dedicated disk drive. [4]

Hardware random number generators are non-deterministic by nature—no algorithm can be used to predetermine subsequent bits. Thus, hardware RNGs are not susceptible to intrusion or exposure by algorithm disassembly or disclosure. The property of non-determinism has been shown to be especially important in specific RNG applications such as certain scientific and financial modeling techniques, government-sponsored lotteries, and computer security technology such as cryptography and digital signatures.

Hardware random number generators do not require seeds because hardware random numbers are not computed values. They are not derived through a repeatable algorithm. Rather, hardware-generated random numbers are digitized snapshots of naturally occurring thermal noise.

Testing for Randomness

There are a variety of tests and benchmarks that can be applied to an RNG to rate its effectiveness. Among these are Dr. George Marsaglia's Diehard tests [2] and the U.S. government's Federal Information Processing Standard FIPS 140-1 specification [5].

The Diehard test suite consists of a battery of statistical tests focused on identifying patterns and non-uniform distribution of numbers within the output bit stream. This test suite, which was originally developed for evaluating the randomness of pseudo random number generators, also attempts to identify short periods of repetition in PRNG output.

The FIPS 140-1 test suite provides measures of four characteristics of RNG output:

- duty cycle (the number of ones divided by total bits generated)
- relative occurrence of 4-bit strings
- runs of consecutive like bits
- absence of runs of 34 or more like bits

Case Study: SSL Security Breach

An incident widely publicized a few years ago illustrates the vulnerability of pseudo random numbers in cryptographic applications. In this incident, the data encryption system of an early version of a Web browser was compromised.

When establishing a secure session, the browser collected data from the system clock and process ID table. It used that data as a seed for a pseudo random number generator, which manipulated the seed and output a pseudo random number. This number was used to create a symmetric key for encrypting and decrypting data through the remainder of the session.

Two graduate students at the University of California at Berkeley discovered that if they had an account on the system on which the browser was running, they could discover the data used to seed the PRNG and guess the key

within one minute. Even without an account on the browser's host system, they were able to reduce the range of possible seed (and, therefore, key) values considerably, making the code relatively easy to break.

The students posted their discovery to a Usenet newsgroup, where it was widely read. When the browser vendor investigated the problem they discovered that the seed data for their PRNG only contained about 30 bits of random data, significantly fewer than necessary to generate strong 40- and 128-bit keys. The vendor quickly addressed the problem by strengthening their PRNG, but not before they received sharp criticism in the press. The perceived magnitude of the security compromise is clear, as quoted from the following article:

> "A serious security flaw has been discovered in . . . software used for computer transactions over the Internet's World Wide Web, threatening to cast a chill over the emerging market for electronic commerce."
>
> -New York Times, September 19, 1995

The weakness in the browser implementation that made it possible to discover keys and decipher messages was a weak system for generating secret keys. Had the browser vendor utilized a hardware random number generator with a true random data source, the encryption mechanism would have been much stronger, and probably would not have been compromised.

Conclusion

A high quality, hardware-based random number generator is absolutely fundamental to providing a high level of information security. Because random numbers are the foundation for secure cryptographic solutions, digital signatures, and protected communication protocols, the best random number generator (RNG) should produce statistically random and indeterministic numbers. Only a hardware RNG can meet both of these requirements. The hardware-based Intel Random Number Generator, integrated into Intel Architecture platforms, strengthens applications like Web browsing, e-Business, and remote access, which currently use software-based random number generators.

References

[1] Davies, Robert. "True Random Numbers." http://webnz.com/robert/true_rng.html (9 Oct. 1998).

[2] "Diehard." http://stat.fsu.edu/~geo/diehard.html (16 Oct. 1998)

[3] Eastlake, D., S. Crocker, and J. Schiller. "Randomness Recommendations for Security." Internet Engineering Task Force RFC 1750, 1994.

[4] Ellison, Carl. "Cryptographic Random Numbers." Draft P1363 Appendix E. http://www.clark.net/pub/cme/P1363/ranno.html (9 Oct. 1998).

[5] FIPS 140-1, "Security Requirements for Cryptographic Modules." Federal Information Processing Standards Publication 140-1. U.S. Department of Commerce/NIST, National Technical Information Service. Springfield, Virginia, 1994. http://csrc.ncsl.nist.gov/fips/fips1401.htm (16 Oct. 1998)

[6] Helsinki University of Technology. "Introduction to Cryptography," http://www.cs.hut.fi/crypto/intro.html (16 Oct. 1998).

[7] Knuth, Donald E. *The Art of Computer Programming.* Vol. 2, *Seminumerical Algorithms*. 2nd ed. Reading, MA: Addison-Wesley, 1981

[8] Markoff, John. "Security Flaw Is Discovered In Software Used in Shopping." *The New York Times*, 19 September 1995, sec. A, p. 1.

[9] Peterson, Ivars. *The Jungles of Randomness*. New York: John Wiley & Sons, 1998.

[10] Schneier, Bruce. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. New York: John Wiley & Sons, 1996.